Kauno technologijos universitetas

Informatikos fakultetas

Projektas

P175B014 Duomenų struktūrų projektinis darbas

Projekto autorius
Gustas Klevinskas

Akademinė grupė
IFF-8/7

Kaunas, 2019

Turinys

Programos idėja

Atvaizduoti smulkius paveiksliukus pasinaudojant sparse matrix duomenų struktūra. Joje bus saugoma informacija apie duomenų vietą (koordinates) ir patys duomenys.

Interface‘as

```
package sparse;

public interface SparseMatrixInterface<T> {

  int size();

  void add(int x, int y, T value);

  T get(int x, int y);

  void remove(int x, int y);

  void clear();

  T[][] toMatrix();
}
```

Interfaceą realizuojanti klasė

```
package sparse;

import java.util.Arrays;
import java.util.Iterator;
import sparse.SparseMatrix.Cell;

public class SparseMatrix<T> implements SparseMatrixInterface<T>,
Iterable<Cell<T>> {

  private Cell<T>[] array;
  private T emptyValue;
  private int size = 0;
  private int initialSize = 10;

  public SparseMatrix(T[][] matrix, T emptyValue) {
    this(emptyValue);

    int matrixLength = matrix[0].length;
    int matrixHeight = matrix.length;

    for (int i = 0; i < matrixLength; i++) {
      for (int j = 0; j < matrixHeight; j++) {
        add(i, j, matrix[i][j]);
      }
    }
  }

  public SparseMatrix(T emptyValue) {
    this.emptyValue = emptyValue;
    array = new Cell[initialSize];
  }

  public SparseMatrix() {
    emptyValue = null;
    array = new Cell[initialSize];
  }

  @Override
  public int size() {
    return size;
  }

  @Override
  public void add(int x, int y, T value) {
    if (!value.equals(emptyValue)) {
      array[size++] = new Cell<>(x, y, value);

      if (size == initialSize) {
        expand();
      }
    }
  }

  @Override
  public T get(int x, int y) {
    for (int i = 0; i < size; i++) {
```

```
    if (array[i].x == x && array[i].y == y) {
      return array[i].value;
    }
  }

  return emptyValue;
}

@Override
public void remove(int x, int y) {
  for (int i = 0; i < size; i++) {
    if (array[i].x == x && array[i].y == y) {
      shift(i);
      break;
    }
  }
}

@Override
public void clear() {
  Arrays.fill(array, null);
  size = 0;
}

@Override
public T[][] toMatrix() {
  int maxX = 0;
  int maxY = 0;

  for (Cell<T> i : array) {
    if (maxX < i.x) {
      maxX = i.x;
    }

    if (maxY < i.y) {
      maxY = i.y;
    }
  }

  @SuppressWarnings("unchecked")
  T[][] matrix = (T[][]) new Object[maxY][maxX];

  for (Cell<T> i : array) {
    matrix[i.y][i.x] = i.value;
  }

  return matrix;
}

private void shift(int index) {
  for (int i = index; i < size - 1; i++) {
    array[i] = array[i + 1];
  }

  array[size] = null;
  size--;
}

private void expand() {
```

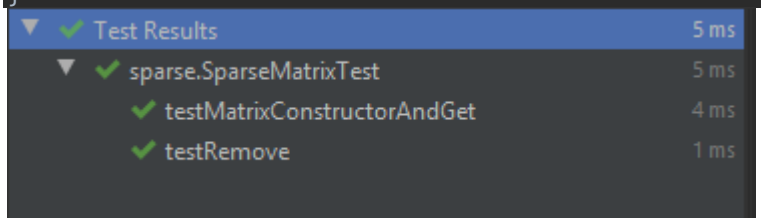
```
    initialSize *= 2;
    Cell<T>[] newArray = new Cell[initialSize];
    System.arraycopy(array, 0, newArray, 0, size);

    array = newArray;
  }

  @Override
  public Iterator<Cell<T>> iterator() {
    return new Iterator<Cell<T>>() {
      int currentIndex = 0;

      @Override
      public boolean hasNext() {
        return currentIndex < size;
      }

      @Override
      public Cell<T> next() {
        return array[currentIndex++];
      }
    };
  }

  public byte[] toBytes() {
    StringBuilder sb = new StringBuilder();
    System.out.println("xd");

    for (int i = 0; i < size; i++) {
      if (array[i] != emptyValue) {
        sb.append(array[i]).append("\n");
      }
    }

    return sb.toString().getBytes();
  }

  public static Cell<?> parseString(String line) {
    String[] vars = line.split("\\|");
    return new Cell<>(Integer.parseInt(vars[0]), Integer.parseInt(vars[1]), null);
  }

  public static class Cell<T> {

    public int x;
    public int y;
    public T value;

    public Cell(int x, int y, T value) {
      this.x = x;
      this.y = y;
      this.value = value;
    }

    @Override
    public String toString() {
      return x + "|" + y + "|" + value;
    }
```

```
    public boolean equals(Cell<T> other) {
      return x == other.x && y == other.y && value.equals(other.value);
    }
  }
}
```

Testavimas naudojant JUnit

```java
package sparse;

import com.sun.tools.javac.util.Pair;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import org.junit.Assert;
import org.junit.BeforeClass;
import org.junit.Test;

public class SparseMatrixTest {

  /**
   * Size of the matrix
   */
  public static final int SIZE = 3;
  public static Integer[][] matrix;
  /**
   * Pairs of integers, that coordinates of not empty cells
   */
  public static List<Pair<Integer, Integer>> coordinates;

  @BeforeClass
  public static void generateMatrix() {
    matrix = new Integer[SIZE][SIZE];
    coordinates = new ArrayList<>();

    for (int i = 0; i < SIZE / 3; i++) {
      int x = new Random().nextInt(SIZE);
      int y = new Random().nextInt(SIZE);
      coordinates.add(new Pair<>(x, y));
    }

    for (int i = 0; i < SIZE; i++) {
      for (int j = 0; j < SIZE; j++) {
        if (coordinates.contains(new Pair<>(i, j))) {
          matrix[i][j] = 69;
        } else {
          matrix[i][j] = 0;
        }
      }
    }
  }

  @Test
  public void testMatrixConstructorAndGet() {
    SparseMatrix<Integer> sparseMatrix = new SparseMatrix<>(matrix, 0);

    for (int i = 0; i < SIZE; i++) {
      for (int j = 0; j < SIZE; j++) {
        if (coordinates.contains(new Pair<>(i, j))) {
          Assert.assertEquals(Integer.valueOf(69), sparseMatrix.get(i, j));
        } else {
          Assert.assertEquals(Integer.valueOf(0), sparseMatrix.get(i, j));
        }
      }
```

```
    }
  }

  @Test
  public void testRemove() {
    SparseMatrix<Integer> sparseMatrix = new SparseMatrix<>(matrix, 0);

    for (Pair<Integer, Integer> i : coordinates) {
      sparseMatrix.remove(i.fst, i.snd);
    }

    Assert.assertEquals(sparseMatrix.size(), 0);
  }
}
```



Greitaveikos tyrimas naudojant JMH

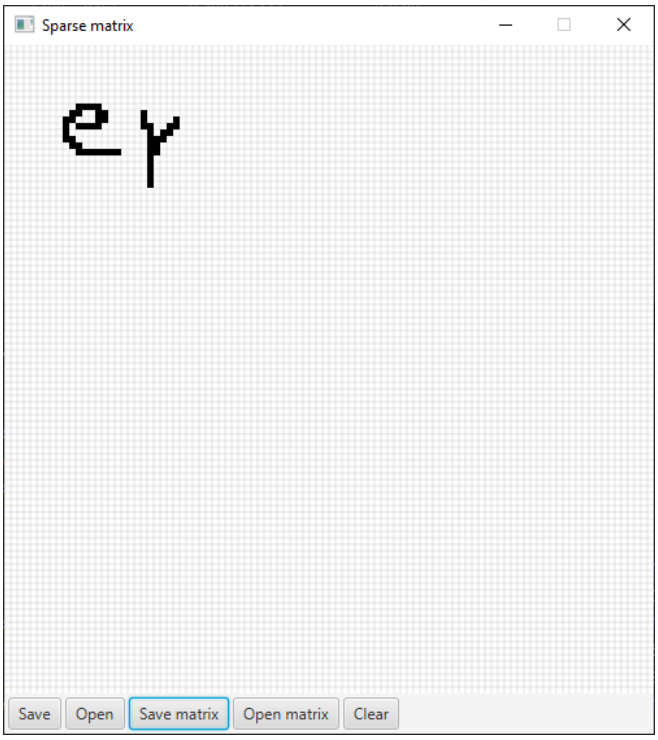
```java
package benchmark;

import java.util.Random;
import java.util.concurrent.TimeUnit;
import org.openjdk.jmh.annotations.Benchmark;
import org.openjdk.jmh.annotations.BenchmarkMode;
import org.openjdk.jmh.annotations.Fork;
import org.openjdk.jmh.annotations.Level;
import org.openjdk.jmh.annotations.Measurement;
import org.openjdk.jmh.annotations.Mode;
import org.openjdk.jmh.annotations.OutputTimeUnit;
import org.openjdk.jmh.annotations.Param;
import org.openjdk.jmh.annotations.Scope;
import org.openjdk.jmh.annotations.Setup;
import org.openjdk.jmh.annotations.State;
import org.openjdk.jmh.annotations.Warmup;
import org.openjdk.jmh.infra.Blackhole;
import sparse.SparseMatrix;
import sparse.SparseMatrix.Cell;

@State(Scope.Benchmark)
@BenchmarkMode(Mode.AverageTime)
@OutputTimeUnit(TimeUnit.NANOSECONDS)
@Fork(value = 1)
@Warmup(iterations = 1)
@Measurement(iterations = 1)
public class SpeedBenchmark {

  @Param({"500", "1000", "1500", "2000"})
  public int matrixSize = 500;

  SparseMatrix<Boolean> sparseMatrix;
  Boolean[][] matrix;

  @Setup(Level.Trial)
  public void generateBoard() {
    Random random = new Random();

    sparseMatrix = new SparseMatrix<>();
    matrix = new Boolean[matrixSize][matrixSize];

    for (int i = 0; i < matrixSize / 4; i++) {
      int x = random.nextInt(matrixSize);
      int y = random.nextInt(matrixSize);

      sparseMatrix.add(x, y, true);
      matrix[x][y] = true;
    }
  }

  @Benchmark
  public void sparseMatrixIterationTest(SpeedBenchmark sb, Blackhole blackhole) {
    for (Cell<Boolean> i : sb.sparseMatrix) {
      blackhole.consume(i);
    }
  }
```

```java
  @Benchmark
  public void matrixIterationTest(SpeedBenchmark sb, Blackhole blackhole) {
    for (int i = 0; i < sb.matrixSize; i++) {
      for (int j = 0; j < sb.matrixSize; j++) {
        if (sb.matrix[i][j] != null) {
          blackhole.consume(sb.matrix[i][j]);
        }
      }
    }
  }
}
```

Benchmark	(matrixSize)	Mode	Cnt	Score	Error	Units
SpeedBenchmark.matrixIterationTest	500	avgt		418032.100		ns/op
SpeedBenchmark.matrixIterationTest	1000	avgt		1697206.345		ns/op
SpeedBenchmark.matrixIterationTest	1500	avgt		3758031.994		ns/op
SpeedBenchmark.matrixIterationTest	2000	avgt		6582061.842		ns/op
SpeedBenchmark.sparseMatrixIterationTest	500	avgt		602.600		ns/op
SpeedBenchmark.sparseMatrixIterationTest	1000	avgt		1180.948		ns/op
SpeedBenchmark.sparseMatrixIterationTest	1500	avgt		1749.173		ns/op
SpeedBenchmark.sparseMatrixIterationTest	2000	avgt		2306.794		ns/op

Sunaudojamos atminties palyginimas

Lyginau faile išsaugotos reprezentacijos dydį tarp SparseMatrix ir paprastos matricos. Saugojau tokį paveiksliuką:

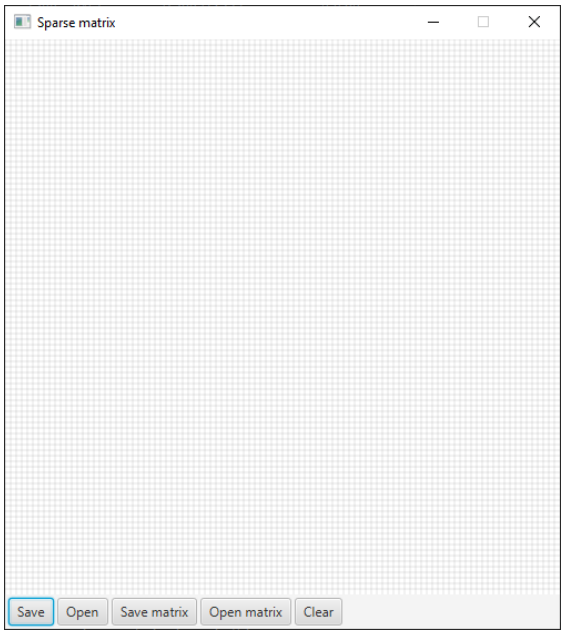


Saugojant SparseMatrix formatu, failas užėmė 553 baitus, tuo tarpu saugojant tokio pat dydžio paprastą boolean reikšmių matricą faile – 9.86 kB. Beveik 20 kartų didesnis failas.

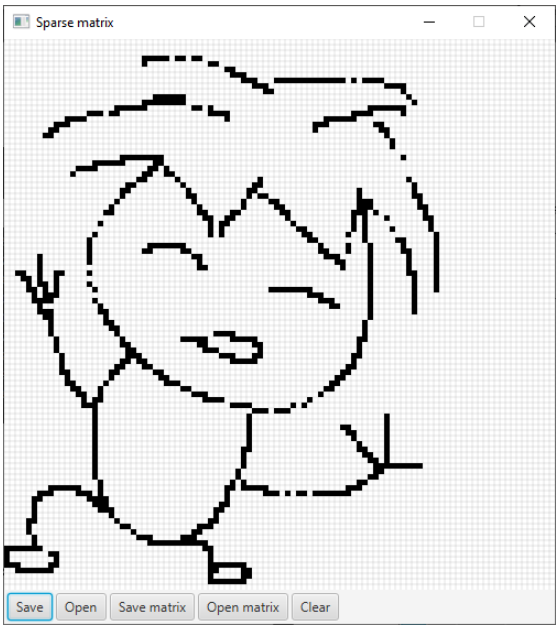
Demo

Suprogramavau programėlę, kurioje galima nupiešti paveiksliuką ir jį išsaugoti specialiame SparseMatrix atitinkančiame faile. Failų dydžio palyginimui dar pridėjau saugojimą pagal paprastą matricą.

Pradžios langas:



Pripaišius vaizdas atrodo taip:



Paspaudus „Save“, paveiksliukas yra išsaugomas projekto aplanke. Esant poreikiui, specialius failus galima peržiūrėti naudojant notepad.

Norint peržiūrėti išsaugotus failus, „Open“ mygtuku atidaromi *.spm (**Sp**arse**M**atrix) failai, o „Open matrix“ atidaromi *.mat (**mat**rix) failai.

Išvados

Projektas buvo atliktas sėkmingai, susipažinau su testų rašymu, bei tikslesnės greitaveikos tyrimo eiga.

SparseMatrix duomenų struktūroje apsimoka saugoti tik mažai užpildytas matricas. Kadangi kiekviena celė SparseMatrix struktūroje turi saugoti koordinates ir reikšmę, esant dideliam elementų kiekiui susidarys daugiau sunaudotos atminties nei paprastoje matricoje.

Iš greitaveikos tyrimo aiškiai matosi, kad praiteruoti pro SparseMatrix yra žymiai greičiau nei per paprastą matricą. Taip yra todėl, kad SparseMatrix saugo tik reikšmę turinčius langelius. Tuo tarpu paprastoje matricoje reikia eiti per kiekvieną langelį, kad paaiškėtų jo reikšmė. Panaudojant SparseMatricą mano sugalvotoje programoje yra išlošiama daug efektyvumo prasme – piešiami tik tie pikseliai, kurie yra nudažyti, ir per kitus net neiteruojama.